On methods for analyzing and improving deep learning

Sørby, Anders Christiansen

March 5, 2019

Abstract

In this project thesis we discuss several approaches to improving deep learning found in research papers. First we give an introduction to deep learning and neural networks. Then we cover some papers that analyze neural networks with information theory and algebraic topology. We also present concepts and results from learning theory like the VC dimension, mixing complexity and the no free lunch theorem. Then finally we present the method Spectral Pruning and do numerical experiments with it.

Contents

1 Introduction

In this project thesis we will discuss a range of topics related to deep learning especially in a theoretical setting. The purpose of this thesis is to motivate further analytic study of deep learning as a mathematical object. This is, among other things, to ensure that the development of methods in deep learning remain safe and understandable. Additionally we note that theoretical results may be useful just as much as empirical ones. We will start with a short introduction to deep learning, then consider how to make networks able to learn more general tasks. After that we will give an introduction to some ways researchers are trying to analyze the theoretical properties of neural networks and finally we will present the technique called *Spectral Pruning* for optimizing a trained network and experiments on the method. This thesis does not intend to be an introduction to deep learning.

1.1 Deep neural networks

We will now give a short introduction to some of the basics of deep learning and neural networks. For a more complete reference we recommend the book *Deep Learning*[GBC16]. Deep learning is a machine learning subfield which is characterized by the use of large models with many layers; hence the name deep. The intention of this is to learn more complicated tasks like image or voice recognition. The quintessential model in deep learning is called a deep neural network (DNN), or just neural network (NN), and is in some sense a chain of linear predictors connected by nonlinear functions. Then we have some input x, which can be for example a color image in the shape of a three-tensor or a text string in the shape of a vector (one-tensor). However there are very many variations to this form and the following form is perhaps the simplest, namely the fully connected NN,

$$f(x) = \mu_L(W_L(\cdot) + b_L) \circ \ldots \circ \mu_1(W_1 x + b_1). \tag{1.1.1}$$

We are also going to use a recursive definition which is very convenient in the Spectral Pruning section. Here the network is represented as a chain of functions (which can also be thought and referred to as links or layers)

$$\begin{aligned} f_1(x) &= \mu_1(W_1 x + b_1), \\ f_\ell(x) &= \mu_\ell(W_\ell(f_{\ell-1}(x)) + b_\ell) \text{ for } \ell = 2, \ldots, L \end{aligned} \tag{1.1.2}$$

where μ_ℓ is some nonlinear activation function working elementwise on input of any dimension and W_ℓ and b_ℓ are the weight matrices and bias vectors respectively. Examples of common nonlinear activation functions are the rectifier $ReLU(x) = \max(x, 0)$ and the sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$. For generality and convenience we are going to denote the parameters (or simply the weights) as ω and ω_ℓ for all parameters or the parameters of a specific layer respectively.

1.2 Additional layer structures

We can imagine replacing one of the layers with another structure that can take the same input and give an output of correct dimension to the next layer. It may even take input from other earlier layers like in residual nets or dense nets. One notable possibility for a layer is a convolutional layer. A convolutional layer can be seen simply as a less connected layer than the fully connected layer where only the local relationships of the input are examined. The concept revolves around doing a convolution operation with some kernel $K_{i,j}$ (also called filters), which corresponds to the weights, for some input x. For example we can consider a two-dimensional kernel, $K_{i,j} \in \mathbb{R}^{N \times M}$, which normally means that the input only has one channel (like a black and white image). This means that the kernel sweeps over the input one neighborhood of size $N \times M$ at a time. We index the kernel with i, j to signify that we can have different kernels for each neighborhood.

$$\begin{aligned} S(i,j) &= (K_{i,j} * x)(i,j) = \sum_{n}^{N} \sum_{m}^{M} x(i-n, j-m) K_{i,j}(n,m) \\ f_\ell(x) &= \mu_\ell(S) \end{aligned} \tag{1.2.1}$$

In this case the output is in the shape of a matrix (two-tensor) not a vector. For three-tensor input (like a color image) the output will also be a three-tensor. To be able to connect this to the next layer one can either flatten the tensor into a vector and possibly lose structural information or keep the structure and let the next layer handle higher order inputs. Also note that again we apply some nonlinear activation function μ_l. A NN containing at least one convolutional layer is usually called a *Convolutional Neural Network* (CNN).

A NN where the output of a layer is sent back as input to an earlier layer and then iterated arbitrarily many times is called a *Recurrent Neural Network* (RNN). This is necessary when you want to produce sequences of arbitrary length as for example in text generation.

1.3 Training

The network can be trained on a dataset $\{(x_i, y_i)\}_{i=1}^{n}$ (or rather a training set), where x_i are called samples and y_i are called labels, such that $f(x_i) = y_i$ for all i or some probability that the sample has a certain label. We can imagine that all samples are collected from a greater sample space $\mathcal{X}$ and all labels from a greater label space $\mathcal{Y}$, and there is a probability distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$. Note that while in general the space of possible NNs and $\mathcal{X}$ and $\mathcal{Y}$ is infinite we sometimes need to restrict ourselves to finite spaces when doing analyzes.

Additionally we have a separate test set $\{(x_i, y_i)\}_{i=n+1}^{n+n_{\text{test}}}$ which we will use to verify the generalization ability of the network. The setting where all the labels of the samples are known is called supervised learning. Training or learning differs from normal optimization in that we wish to optimize the performance on the test set indirectly by optimizing the performance on the training set. We can not know anything about the generalization ability of the network without a separate test set which has not been used in training. An important note from this is that when we are learning we do not know the underlying distribution of data.

For simplicity and notational convenience we will sometimes use slightly different versions of the loss function. This will be indicated with different indexes on the function or different arguments to the function. Remember that we denoted the parameters of the network as ω. To train the network we first need to compute some loss function $\mathcal{L}(\omega, x_i, y_i)$, which in this case applies to a single sample. This will give a score of the accuracy of the network on a given sample. The training problem then becomes to minimize the loss

$$\min_{\omega} \mathcal{L}(\omega, x_i, y_i) + R(\omega) \tag{1.3.1}$$

where $R(\omega)$ is some optional regularizer. A popular choice for loss function is the cross entropy, which compares the entropy of two probability distributions, and follows from the maximum likelihood principle. In this case the empirical distribution from the training set and the prediction distribution from the model is compared

$$\mathcal{L}(\omega) = -\mathbb{E}_{(x,y)\sim\hat{\mathcal{D}}_{\text{data}}}\left[\log(p_{\text{model}}(y|x))\right]. \tag{1.3.2}$$

This way we reduce a machine learning problem to a normal optimization problem by introducing the empirical distribution $\hat{\mathcal{D}}_{\text{data}}$. There are however many other candidates for loss functions.

Then to update the weights accordingly it is normal to use a variant of *Stochastic Gradient Descent* (SGD) and back propagation to compute the gradient for each layer. It requires that the total loss over all training samples, $\mathcal{L}_{\text{total}}(\omega, \{x_i\}, \{y_i\})$, is an average over the loss for each individual sample. This means that in general it follows the form

$$\mathcal{L}_{\text{total}}(\omega, \{x_i\}, \{y_i\}) = \frac{1}{n}\sum_i \mathcal{L}(\omega, x_i, y_i) \tag{1.3.3}$$

$$\omega \leftarrow \omega - \eta\nabla\mathcal{L}(\omega, x_i, y_i) \tag{1.3.4}$$

where the hyperparameter η is called the learning rate. This is then performed iteratively over the entire training set, where one iteration over the training set is called an epoch.

1.4 Limits to learnability

There is a limit to which data that can be learned by a certain network, and training a NN has even been shown to be NP-complete [BR88]. This difficulty can be characterized by a complexity measure. In traditional statistical learning theory PAC learnability and the VC-dimension are central to classifying the limitations of a learning algorithm. Given an underlying true hypothesis f and a distribution $\mathcal{D}$ over samples the test error, or empirically the loss, is the probability that our proposed hypothesis, h, is wrong $\mathcal{L}_{f,\mathcal{D}}(h) = P_{x\sim\mathcal{D}}[h(x) \neq f(x)]$. The definitions are restricted to binary classification for simplicity.

A way to analyze the potential of a learning method is to consider it as a hypothesis class, $\mathcal{H}$, which contains all the different functions, called hypotheses, realizable by the method. However this says nothing about how to learn a specific function and the memory and computational cost (time) of running the algorithm. Therefore even if a hypothesis is contained in $\mathcal{H}$ it may not be feasible to learn it. Restrictions to memory and computation can be added later.

Definition 1.4.1. *Probably Approximately Correct (PAC) learnability is for a hypothesis class $\mathcal{H}$ the existence of an algorithm, $\mathcal{A}_{\mathcal{H}}$, and a sample limiting function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ such that for every*

- $\epsilon, \delta \in (0,1)$
- *Distribution $\mathcal{D}$ over $\mathcal{X}$*
- *underlying perfect hypothesis or true function $f : \mathcal{X} \to \{0,1\}$*

given $m \geq m_{\mathcal{H}}(\epsilon, \delta)$ i.i.d training samples from $\mathcal{D}$ the algorithm produces a hypothesis $h \in \mathcal{H}$ such that, with probability $1-\delta$ the test error is

$$\mathcal{L}_{f,\mathcal{D}}(h) \leq \epsilon.$$

Next we introduce the more traditional measure of complexity, the VC dimension, and the related concepts.

Definition 1.4.2. *A **restriction** of a hypothesis class $\mathcal{H}$ in the domain $\mathcal{X}$ into a set $S = \{x_1, \dots, x_n\} \subseteq \mathcal{X}$ is the set of tuples of S over every hypothesis h in $\mathcal{H}$,*

$$\mathcal{H}_S = \{h[S] = (h(x_1), \dots, h(x_n)) | h \in \mathcal{H}\}$$

Let $f[S] = (f(x_1), \dots, f(x_n))$ be the tuple of S over some true function f.

Definition 1.4.3. *A hypothesis class $\mathcal{H}$ **shatters** a set S if for any true function f, $f[S]$ is in $\mathcal{H}_S$, or equivalently $|\mathcal{H}_S| = 2^n$.*

Intuitively shattering can be thought of as $\mathcal{H}$ being able to completely understand S.

Definition 1.4.4. *The Vapnik–Chervonenkis (VC) dimension of a hypothesis class $\mathcal{H}$ is the cardinality of the biggest set S that $\mathcal{H}$ shatters,*

$$VC(\mathcal{H}) = \sup\{|S| \mid \mathcal{H} \text{ shatters } S\}.$$

Theorem 1.1 (The Fundamental Theorem of Statistical PAC Learning)**.** *For a hypothesis class $\mathcal{H}$ over $\mathcal{X}$, with $VC(\mathcal{H}) = d$, there are constants U and L such that $\mathcal{H}$ is PAC learnable with sample complexity (requirement)*

$$L\frac{d - \log\delta}{\epsilon} \leq m_{\mathcal{H}}(\epsilon, \delta) \leq U\frac{-d\log\epsilon - \log\delta}{\epsilon}$$

for every $\epsilon, \delta \in (0, 1)$.

Proof and a more complete version of the theorem can be found in the book [SSBD14]. This theorem establishes the implications the VC-dimension has for the potential of a learning method, i.e. how many samples it takes to train the network to a certain accuracy. What we want is to have a method that requires the least amount of samples to perform well. That means that we would want the VC-dimension to be big. Note that the VC-dimension is problem dependent so comparison of different methods on different problems is not fair.

However this may not be as explanatory as we would like. In the paper [MT17] they introduce a complexity measure called *Mixing Complexity* which explores the ability of bounded memory algorithms to learn. It requires that we represent the hypothesis class as a bipartite graph which is a graph were all the nodes can be separated into two groups, specifically hypotheses $h \in \mathcal{H}$ and examples $x \in \mathcal{X}$, where there is no edges inside the groups. Then there is an edge $(h, x) \in E$ if and only if $h(x) = 1$.

Definition 1.4.5. *A hypothesis class $\mathcal{H}$ is, for an integer $d > 0$, **d-mixing** if, represented as a bipartite graph $G = (\mathcal{H}, \mathcal{X}, E)$, for every $T \subseteq \mathcal{X}$ and $S \subseteq \mathcal{H}$ the following relation holds*

$$\left| e(S, T) - \frac{|S||T|}{2} \right| \leq d\sqrt{|S||T|}$$

where $e(S, T)$ is the number of edges between the two node sets.

Definition 1.4.6. *For a hypothesis class $\mathcal{H}$ over $\mathcal{X}$ with minimal d-mixing $d_{min}(\mathcal{H}) = \min_d\{\mathcal{H}$ is d-mixing$\}$ the **mixing complexity** is*

$$MC(\mathcal{H}) = \frac{\sqrt{|\mathcal{H}||\mathcal{X}|}}{d_{min}(\mathcal{H})}$$

We say that a class is *mixing* if $MC(\mathcal{H}) = \Omega(\sqrt{|\mathcal{H}|})$. We also know that any memory bounded learning algorithm cannot learn a mixing hypothesis class from theorem 8 in [MT17]. By cannot learn we mean that any memory bounded algorithm would require at least $|\mathcal{H}|^c$ training examples for some small constant $c > 0$. For example the class of thresholds

$$\mathcal{H}_{th} = \{h_b : \mathcal{X} \to \{0,1\} \mid h_b(x) = 1 \iff x < b \in [0,1]\}$$
$$\mathcal{X} = \left\{ \frac{i}{|\mathcal{X}| - 1} \,\middle|\, i = 0, \dots, |\mathcal{X}| - 1 \right\}$$

has $MC(\mathcal{H}_{th}) = O(1)$ which means that it is mixing.

This way of measuring the complexity accounts for the fact that most learning algorithms for NNs are memory bounded. It is known that an unbounded algorithm can learn on $O(\log|\mathcal{H}|)$ examples.

2 Understanding deep models with a mathematical toolbox

If deep learning is going to be applied to practical areas where the output needs to be reliable we need to ensure that their behaviour and properties are well understood. For example if an image recognition algorithm is used in a self driving car getting the correct answer could mean life or death. Relying too much on empirical guaranties can be dangerous. This especially considering the problem of adversarial examples in deep learning which is described and analyzed in this paper [WLT+18]. Basically it is possible to do specific changes on data points that gives a behaviour very different than what is desired. An attacker could potentially trick a DNN to do specific actions by presenting data points that are engineered for that purpose. This even applies to the real world when an image classifier is presented with real world images that it classifies wrongly but the changes are unnoticeable for humans. This motivates the need for a thorough theoretical analysis of deep learning such that we can have rigorous understanding of deep learning methods.

Because of the large complexity of DNNs theoretical properties are hard to study using conventional methods from machine learning and statistics. Here we are going to consider the application of two mathematical theories namely information theory and algebraic topology.

2.1 Mutual information

The properties and the resulting success of NNs is hard to analyze analytically and therefore elusive. The paper [ST17] gives a theoretical analysis of NNs through an information theoretic lens. Most importantly one can identify two distinct phases of training in terms of information. A good reference for information theory is the book *Elements of information theory*[CT06] The entropy, $H(X)$, and shared information, $I(X;Y)$, of two random variables are of interest

$$H(X) = -\mathbb{E}[\log(p_X(X))] = -\sum_{x\in X} p_X(x)\log p_X(x) \tag{2.1.1}$$

$$H(X|Y) = -\sum_{x\in X, y\in Y} p_{X,Y}(x,y)\log\left(\frac{p_{X,Y}(x,y)}{p_Y(y)}\right) \tag{2.1.2}$$

$$\begin{aligned} I(X;Y) = -\sum_{x\in X, y\in Y} p_{X,Y}(x,y)\log\left(\frac{p_X(x)p_Y(y)}{p_{X,Y}(x,y)}\right) &= D_{KL}[p_{X,Y}(x,y)||p_X(x)p_Y(y)] \\ =& H(X) - H(X|Y) \end{aligned} \tag{2.1.3}$$

where p_X is the probability density function for the given index, $H(X|Y)$ is the conditional entropy and $D_{KL}[p||q]$ is the Kullback-Leibler divergence.

A key component of this analysis is the concept of the *Information Bottleneck* (IB) [TPB99]. The concept builds on optimal representations of some data X with respect to some related data Y. We will treat this data as random variables over some distribution. Classically this is solved with a sufficient statistic $S(X)$, which are maps or partitions of X, such that the information is preserved $I(S(X);Y) = I(X;Y)$. Next we can optimize this representation to obtain a minimal sufficient statistic, $T(X)$, which encodes X in the simplest way. This can be reduced a constrained optimization problem as follows

$$T(X) = \arg\min_{S(X)} I(S(X);X) \text{ such that } I(S(X);Y) = I(X;Y). \tag{2.1.4}$$

Exact solutions of this problem rarely exists. However we can relax the problem by allowing the map to be stochastic, which can be considered as an encoder $P(T|X)$, and allowing the map to capture as much information as possible rather than exactly everything. Let $t \in T$ be the compressed representations of $x \in X$ which gives the mapping $p(t|x)$. This is more apparent if we consider this representation operation as a Markov chain $Y \to X \to T$. Then we can formulate the IB tradeoff via an optimization problem over the distribution of representations, $p(t)$, the distribution of representations of x, $p(t|x)$, and the distribution of the information we want to represent $p(y|t)$ which results in

$$\min_{p(t|x),p(y|t),p(t)} I(X;T) - \beta I(T;Y) \tag{2.1.5}$$

where β is a Lagrange multiplier determining the level of information captured by the representation. The solution of this is given by three equations

$$\begin{cases} p(t|x) &= \frac{p(t)}{Z(x;\beta)} \exp(-\beta D_{KL}[p(y|x)||p(y|t)]) \\ p(t) &= \sum\limits_{x\in X} p(t|x)p(x) \\ p(y|t) &= \sum\limits_{x\in X} p(y|x)p(x|t). \end{cases} \tag{2.1.6}$$

Here $Z(x;\beta)$ is the normalization function which is forcing the probability to sum to one. This generates an *information curve* of optimal representations separating the achievable and unachievable regions of the information plane. It is also necessary to add some noise to the encoding rule $p(t|x)$ to make it stochastic. Otherwise the mutual information can not distinguish low complexity classes (low VC dimension) from high complexity classes (high mixing complexity).

This tool can then be used to analyze the mutual information of a DNN in the information plane which they do in [ST17]. To calculate the mutual information .

Important results are that the training can be separated into two distinct phases. First there is the *drift phase* where SGD increases the $I(T_i;Y)$. Then when the training error begins to become small there is the *representation compression phase*.

2.2 Maximum entropy

In the paper [ZSX17] they develop a connection between maximum entropy and deep learning building on the information bottleneck (IB) principle discussed earlier. They establish an alternative perspective on the results found in [ST17] with maximum entropy as the basis.

The maximum entropy principle is a general principle designed to minimize the generalization error. It can be summarized as an optimization problem as follows

$$\max_{\hat{Y}} H(\hat{Y}|X) \tag{2.2.1}$$
$$\text{Such that: } P(X,Y) = P(X,\hat{Y}), \text{ and}$$
$$\sum_{\hat{Y}} P(\hat{Y}|X) = 1$$

where $\hat{Y}$ is a prediction; the output of a model. Essentially the maximum entropy predictor is expected to generalize better. This is because it has the minimal number of extra hypotheses on the data.

2.3 Neural homology

The following section assumes some knowledge of group theory and only gives a short introduction to algebraic topology which can be studied in detail in the book *Algebraic topology*[Hat02]. Topological data analysis (TDA) tries to compute the topological features of data. Algebraic topology gives a new perspective to data since it roughly considers what is preserved through continuous deformations. When the data are points in a high dimensional space we need a method to extract topological features.

A tool to analyze the topology of a space is homology. It considers the space through homology groups $H_n(\mathcal{X})$. Specifically $H_n(\mathcal{X}) \cong \mathbb{Z}^{\beta_n}$ where β_n is called the nth Betti number and can be considered as the number of n-dimensional holes in $\mathcal{X}$. For example β_0 is the number of connected components of $\mathcal{X}$. The entire homology is the collection of these groups. It can be constructed from a chain complex of groups $C_n(\mathcal{X})$ encoding information about $\mathcal{X}$. They are connected with a boundary map $\partial_n : C_n(\mathcal{X}) \to C_{n-1}(\mathcal{X})$. These maps follow the rule $\partial_n \circ \partial_{n+1} = 0$ where 0 represents a map to the trivial group.

$$\cdots \xrightarrow{\partial_{n+1}} C_n \xrightarrow{\partial_n} C_{n-1} \xrightarrow{\partial_{n-1}} \cdots \xrightarrow{\partial_2} C_1 \xrightarrow{\partial_1} C_0 \xrightarrow{\partial_0} 0 \tag{2.3.1}$$

Then the homology groups are then defined as

$$H_n(\mathcal{X}) := \ker(\partial_n)/\mathrm{im}(\partial_{n+1}) = Z_n(\mathcal{X})/B_n(\mathcal{X}) \tag{2.3.2}$$

where $\ker(\partial_n)$ and $\mathrm{im}(\partial_n)$ is the kernel and image of ∂_n respectively and / means that we are taking the quotient which has some intuitive similarity with division.

A technique that can be used is called *Persistent Homology* [ZC05] which roughly looks at homology of the data at different resolutions. It considers filtrations of the sample space which is a sequence of contained subspaces $\mathcal{X}_0 \subset \mathcal{X}_1 \subset \ldots \subset \mathcal{X}$. In particular it creates a ball of radius ε around each data point. We can consider the data points as a simplicial complex, which is a generalized graph, were initially all the points are unconnected. We then let ε grow such that all the balls grow bigger. When two balls start to intersect we draw an edge between the two points. Eventually all the points of the space becomes connected. We can then plot the life cycle of each component as ε grows. Basically, looking at a specific Betti number say β_0, we draw a line for each component from when it emerges until it is made a part of another component as ε grows. This creates a plot which is called for its visual similarity the *persistence barcode diagram* of $\mathcal{X}$.

In the paper [GS18] they analyze the persistent homology of image data and show that the topological features of a NN limits its ability to generalize. They show by experiment that a network with too few hidden units cannot express all the topological holes in the data and therefore fails to learn.

They also establish a way to characterize the architecture of a NN with homology. Let $f : \mathcal{X} \to \{0,1\}$ be a binary classifier, and $\mathcal{X} \times \{0,1\}$ be a labeled topological space with a distribution $\mathcal{D}$. Then let $H_S(f) := H[f^{-1}(\{1\})]$ be the *support homology* of f. The idea is that given a dataset $\mathcal{T} \subset \mathcal{X} \times \{0,1\}$ the architectures A, with a corresponding class of NNs $\mathcal{F}_A$, can learn $\mathcal{T}$ if there exists a $f \in \mathcal{F}_A$ such that $H_S(f) = H(\mathcal{T})$. The formal version of this can be seen in theorem 3.1 in [GS18]. We can consider neural homology as another complexity measure similar to VC dimension and mixing complexity, though not necessarily associated with a hypothesis class.

3 The road to more general learning

How do we create algorithms that can learn more general tasks? In the paper [KGS+17] they manage learn 8 different task, of different modality, to a single network, MultiModel. The tasks include for instance translation, image recognition, image captioning and speech recognition. However it is not really a single model. Instead there are separate input networks and output networks for each task or modality. The input networks are connected by a general encoding network that is used for all the tasks. This shows that it is possible to have more general networks that can process some general features. It also takes into account the no free lunch theorem since it is not trying to learn a general model for all the tasks, which we will show is somewhat futile, but take advantage of similarities in the data processing of each task.

3.1 The no free lunch theorem

First we will look at some theoretical results on generality of machine learning. The no free lunch theorem indicates that there is no general learning algorithm that performs better on every possible problem. The version presented here is mostly as presented in the book [SSBD14], but generalizes the bound on the loss. Note that in this section we omit the function index from the loss function for simplicity.

Theorem 3.1 (No free lunch)**.** *Let $\mathcal{A}$ be any learning algorithm for binary classification, with respect to the 0-1loss, over some domain $\mathcal{X}$. Then for any training set $\mathcal{T} \subset \mathcal{X} \times \{0,1\}$ with $|\mathcal{T}| = m \leq \frac{|\mathcal{X}|}{2}$ there exists a distribution $\mathcal{D}$ over the labeled domain $\mathcal{X} \times \{0,1\}$ such that*

1. *There exists a function $f : \mathcal{X} \to \{0,1\}$ where $\mathcal{L}_{\mathcal{D}}(f) = 0$.*
2. *Given some $a \in (0, 1/4)$ and $b = \frac{1/4-a}{1-a}$ bounding the expected loss, with probability at least b over the choice of $\mathcal{T} \sim \mathcal{D}^m$, we have $\mathcal{L}_{\mathcal{D}}(\mathcal{A}(\mathcal{T})) \geq a$.*

So there will always be a function that the algorithm fails to learn, but still can be learned by some other method. This is modified version of the more general original theorems from [WM97] where they show that for any optimization problem the average performance of any optimization algorithm over all problems is independent of the algorithm. The proof of the theorem gives more insight into its properties.

Proof. Let $C \subset \mathcal{X}$ be a subset of size $2m$. There are then $N = 2^{2m}$ possible functions $\{f_i | f : C \to \{1,0\}\}_{i=1}^{N} = 2^C$. We can then assign a distribution $\mathcal{D}_i$ over $C \times \{0,1\}$ to each function

$$\mathcal{D}_i(\{(x,y)\}) = \begin{cases} \frac{1}{|C|} & \text{if } y = f_i(x) \\ 0 & \text{otherwise.} \end{cases} \tag{3.1.1}$$

So for this distribution all the samples with correct label for f_i will be uniformly distributed and the incorrect labeled samples has probability 0. This means that $\mathcal{L}_{\mathcal{D}_i}(f_i) = 0$, which will give us the first statement.

Next for any learning algorithm, $\mathcal{A}$, that takes in a training set of m examples from $C \times \{0,1\}$ and outputs a function $A(\mathcal{T}) = h_{\mathcal{T}} : C \to \{0,1\}$, we will show that it holds that

$$\max_{i \in [N]} \mathbb{E}_{\mathcal{T} \sim \mathcal{D}_i^m}[\mathcal{L}_{\mathcal{D}_i}(h_{\mathcal{T}})] \geq \frac{1}{4}. \tag{3.1.2}$$

Then we know that more generally for every algorithm, $\mathcal{A}'$, there exists a function, f, and a distribution, $\mathcal{D}$, with $\mathcal{L}_{\mathcal{D}}(f) = 0$, and a corresponding training set $\mathcal{T} \subset \mathcal{X} \times \{0,1\}$ such that

$$\mathbb{E}_{\mathcal{T} \sim \mathcal{D}^m}[\mathcal{L}_{\mathcal{D}}(\mathcal{A}'(\mathcal{T}))] \geq \frac{1}{4}. \tag{3.1.3}$$

Essentially there is always a distribution that $\mathcal{A}'$ fails to learn on.

We can now show the second statement $P(\mathcal{L}_{\mathcal{D}}(\mathcal{A}(\mathcal{T})) \geq a) \geq b$. Let $\mathcal{D}$ simply be the worst case distribution from (3.1.2). Then note that the loss can be considered as a random variable $\theta := \mathcal{L}_{\mathcal{D}}(\mathcal{A}(\mathcal{T}))$ and we can conveniently choose $a, b \in (0, 1/4)$ such that

$$\mathbb{E}[\theta] = \mu \geq \frac{1}{4} = b + a - ba. \tag{3.1.4}$$

A possible choice of a could be $1/8$ then $b = \frac{1/4 - a}{1-a} = 1/7$.

Then we are going to use using lemma B1 in [SSBD14], which is a consequence of Markov's inequality, and in our case $P(\theta \geq a) \geq \frac{\mu - a}{1-a} \geq b$, which is the statement we want to prove.

What remains to prove is (3.1.2). There are $K = (2m)^m$ possible sequences of size m from C, which we will denote $S_1, \dots, S_K$. Then for each such set there is a training set labeled by f_i, $\mathcal{T}_j^i = \{(x, f_i(x)) | \forall x \in S_j\}$. If we have the distribution $\mathcal{D}_i$ the probability of sampling any training set is the same so

$$\mathbb{E}_{\mathcal{T} \sim \mathcal{D}_i^m}[\mathcal{L}_{\mathcal{D}_i}(\mathcal{A}(\mathcal{T}))] = \frac{1}{K} \sum_{j=0}^{K} \mathcal{L}_{\mathcal{D}_i}(\mathcal{A}(\mathcal{T}_j^i)). \tag{3.1.5}$$

Now we can bound the worst case expectation over all functions, namely

$$\max_{i \in [N]} \mathbb{E}_{\mathcal{T} \sim \mathcal{D}_i^m}[\mathcal{L}_{\mathcal{D}_i}(\mathcal{A}(\mathcal{T}))] \geq \frac{1}{N} \sum_{i=1}^{N} \frac{1}{K} \sum_{j=1}^{K} \mathcal{L}_{\mathcal{D}_i}(\mathcal{A}(\mathcal{T}_j^i)) \tag{3.1.6}$$

$$\geq \min_{j \in [K]} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{\mathcal{D}_i}(\mathcal{A}(\mathcal{T}_j^i)). \tag{3.1.7}$$

Here we used that the average is smaller than the maximum and that the minimum is smaller than the average. Next we will examine the form of the loss for some particular j. Since we are in a finite setting we are going to use the indicator function which is defined like this for some statement Q

$$I(Q) = \begin{cases} 1 & Q \text{ is true} \\ 0 & Q \text{ is false} \end{cases}$$

to clarify counting the elements of subsets. Let $\{v_1, \dots, v_p\} = V_j \subset C$ where $v_r \notin S_j \forall r$ be the examples unknown to the algorithm, then for every function $h : C \leftarrow \{0, 1\}$

$$\mathcal{L}_{\mathcal{D}_i}(h) \geq \frac{1}{2m}|\{x \in C | h(x) \neq f_i(x)\}| \tag{3.1.8}$$
$$= \frac{1}{2m} \sum_{x \in C} I(h(x) \neq f_i(x)) \tag{3.1.9}$$
$$\geq \frac{1}{2m} \sum_{x \notin S_j} I(h(x) \neq f_i(x)) \tag{3.1.10}$$
$$\geq \frac{1}{2p} \sum_{r=1}^{p} I(h(v_r) \neq f_i(v_r)). \tag{3.1.11}$$

Now we can continue to bound the expectation for our output function

$$\frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{\mathcal{D}_i}(\mathcal{A}(\mathcal{T}_j^i)) \geq \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2p} \sum_{r=1}^{p} I(\mathcal{A}(\mathcal{T}_j^i)(v_r) \neq f_i(v_r)) \tag{3.1.12}$$
$$= \frac{1}{2} \cdot \frac{1}{p} \sum_{r=1}^{p} \frac{1}{N} \sum_{i=1}^{N} I(\mathcal{A}(\mathcal{T}_j^i)(v_r) \neq f_i(v_r)) \tag{3.1.13}$$
$$\geq \frac{1}{2} \cdot \min_{r \in [p]} \frac{1}{N} \sum_{i=1}^{N} I(\mathcal{A}(\mathcal{T}_j^i)(v_r) \neq f_i(v_r)). \tag{3.1.14}$$

We can then partition all the possible functions f_i into pairs which disagree on every point outside the training set namely $\{(f_i, f_{i'}) | f_i(x) \neq f_{i'}(x) \iff x \in V_j\}$, which will have size $N/2$. The produced training sets for these pairs will be the same, $\mathcal{T}_j^i = \mathcal{T}_j^{i'}$. It follows that one function in the pair will always agree with the true function, that is

$$I\left[\mathcal{A}(\mathcal{T}_j^i)(v_r) \neq f_i(v_r)\right] + I\left[\mathcal{A}(\mathcal{T}_j^{i'})(v_r) \neq f_{i'}(v_r)\right] = 1. \tag{3.1.15}$$

This means that our sum over possible labeling functions is

$$\frac{1}{N} \sum_{i=1}^{N} I(\mathcal{A}(\mathcal{T}_j^i)(v_r) \neq f_i(v_r)) = \frac{1}{N} \cdot \frac{N}{2} = \frac{1}{2} \tag{3.1.16}$$

Now we can complete expression (3.1.14) and we obtain 1/4 which is what we wanted to prove in (3.1.2). This concludes the proof. □

An intuitive conclusion of this theorem is that if an algorithm performs better than random search on a class of problems it will perform worse on all other problems. Whether those problems are interesting or not is not known, but this means that choosing the proper strategy of learning for a specific problem is essential.

3.2 Deep reinforcement learning

In many cases a labeled data set is difficult or not feasible to obtain. One way to solve this is to use unsupervised learning to label the data or some hybrid like semi supervised learning. However we do not necessarily need to label the data explicitly. If we rather can design a reward function that can reinforce desirable behaviour. This is called reinforcement learning, or deep reinforcement learning when applied to deep learning, and is summarized nicely in [Li17].

3.3 Generative adversarial networks

One model particularly worthy of a mention the Generative Adversarial Networks (GAN) first introduced in [GPAM+14]. It consists of two DNNs, the generator g and the discriminator d, playing a zero sum minimax game which has the formulation

$$\min_g \max_d \left\{ V(d,g) = \mathbb{E}_{x\sim\mathcal{D}_{\text{data}}}[\log d(x)] + \mathbb{E}_{z\sim\mathcal{D}_z}[\log(1-d(g(z)))] \right\}. \tag{3.3.1}$$

Here $V(g,d)$ can be interpreted as the joint loss function for the generator and the discriminator. This is very useful because the generator learns the distribution of the data. It essentially learns to sample new versions of the data making it possible to use much fewer examples in training. For example given a dataset of images the generator can generate new images similar to the data. This also means that the model is not dependent on having labeled examples; it can do unsupervised learning.

There is also an improved version [ACB17] which is using the earth mover (EM), or Wasserstein-1, metric

$$W(p_g,p_r) = \inf_{\mathcal{D}\in\Pi(p_r,p_g)} \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\|x-y\|\right] \tag{3.3.2}$$

where $\Pi(p_r,p_g)$ is the set of all joint probabilities that has p_r and p_g as marginal distributions respectively. This metric induces a weaker topology on the space of probability distributions than for example the Kullback-Leibler divergence og Jensen-Shannon divergence. This ensures that given a locally Lipschitz continuous mapping f_θ (for example a NN) $W(p_r,p_\theta)$ is continuous everywhere and differentiable almost everywhere. This gives rise to the Wasserstein GAN (WGAN). It tackles the problem that training GANs can be difficult because of failure to converge and local minima.

There is also a paper where they use functional gradients to improve the training of WGANs [NS18]. Essentially they note that the generator may have to little representative power and expands the problem to infinite dimensional spaces. This is realized as a gradient layer that through functional gradient descent minimizes the loss in the forward propagation of the network. Let $g = g_2 \circ g_1$ be a generator NN and d be a discriminator NN, then we want to find an intermediate function ϕ that minimizes the loss; essentially

$$\phi = \arg\min_{\phi'} \mathcal{L}_d(g_2 \circ \phi' \circ g_1) \tag{3.3.3}$$

where $\mathcal{L}_d(g) = \mathbb{E}_{z\sim\mu_n}[-f(g(z))]$ and μ_n is the distribution of the noise fed into the generator. The procedure of finding ϕ defines the *gradient layer.*

3.4 Explaining predictions using LIME

Machine learning models and predictions in general encounters an additional complication when applied to real world scenarios - why is a particular prediction reliable? If a model is going to be applied to situations where there is risk involved, like for example in medicine, it is necessary that the model not only can give a accurate prediction, but also explaining which features that made it come to that conclusion. Otherwise people will feel uncomfortable using it and errors and accidents may occur. To have humans able to supervise the predictions of a model can be essential for its applicability.

A recently developed method for this is Local Interpretable Model-agnostic Explanations (LIME) [RSG16]. The concept revolves around training an explainable classifier, $g \in G$, locally around some prediction point. The class G can for example be of linear models. Then let $\pi_x(z)$ be a proximity measure around a data point x. This defines the locality around x; essentially which samples that are similar to x. Next let $\mathcal{L}(f, g, \pi_x)$ be a measure of how unfaithful g is at approximating f, the model we want to explain, in the locality defined by π_x. We also need to ensure that the explaining model g is as simple and interpretable as possible. This can be done by introducing a complexity measure $C(g)$ that gives a numerical representation of model complexity; for example the number of non-zero parameters in a linear model. The result of this is that we want to find the best interpretability-fidelity trade-off for g which gives the problem

$$\xi(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + C(g). \tag{3.4.1}$$

This enables an additional model check and can uncover problems with the model and dataset that is not possible to see purely from test scores or raw data. For example in [RSG16] they test their method on an SVM classifying news articles. Even though the model has 94% accuracy the explanation uncovers that the model uses features that are actually irrelevant for the classification, but happens to be more frequent in one of the classes.

3.5 Automatic model design

Deep learning is dependent on a well designed architecture. Designing the architecture of a DNN requires expert knowledge in the particular field of data; especially if you want state of the art performance. Finding the architecture best suited for learning a particular dataset and also capable of being deployed to edge devices in a scalable manner is not practical. This motivates the need for automatic model design which searches through an architecture space given a dataset and outputs a DNN trained on that dataset. Here we will discuss two promising candidates namely *Efficient Neural Architecture Search* (ENAS) [PGZ+18] and *NASNet* [ZVSL17] .

ENAS searches for an optimal subgraph of a computational graph of the network. It builds on Neural Architecture Search (NAS) which can be summarized as follows. There is a controlling RNN which will guide the search. The RNN first samples an architecture, that is a candidate child DNN, and then trains it to convergence to determine its accuracy. This is then done iteratively until one obtains a model with the best accuracy. The main innovation of ENAS is that it optimizes the performance by letting the candidate child DNNs share parameters through training. This is motivated by the success of transfer learning[TSK+18] which is a way of transferring knowledge between DNNs trained on somewhat similar tasks.

NASNet uses the idea of determining "cells" or convolutional layers on a smaller dataset and the transferring that knowledge on architecture to a larger network. The key component of this is the NASNet search space that allows for transferability in that the complexity of the architecture is independent of depth and input size. In their experiments they first train on the smaller CIFAR-10 dataset and then transfer that knowledge to a model for the much larger ImageNet dataset. Both the small and the large model achieve state-of-the art performance on their respective datasets; better than any human designed model architecture.

4 Spectral pruning

Deep neural networks can be used for many applications, but their size might often be unnecessarily large. It is therefore useful for performance to compress their size to only a strictly necessary number of parameters. The technique spectral pruning described in [SAM+18] addresses this problem by analyzing the generalization error. We will introduce the theory and replicate the technique in that paper.

The technique requires the network to be already trained on some task. Preferably the network has too many parameters so that there is actually some parameters to remove. On the other hand it will not work on a network with too few parameters.

Also note that we are solving a problem similar to automatic model design. So perhaps insights from spectral pruning can be used to improve those methods as well.

4.1 Derivation of algorithm

Given a network similar to (1.1.1) the architecture is determined by the choice of the number of layers L, the dimension of the layers $\{m_\ell\}_{\ell=1}^L$ and the activation function μ_ℓ. For this report we will focus on the choice of $\{m_\ell\}_{\ell=1}^L$.

This method uses the distribution of output, given some data, from each layer to determine the *degree of freedom*. We will consider one layer, ℓ, at a time so we can omit unnecessary indexes for simpler notation. First let $\phi(x) = f_{\ell-1}(x) \in \mathbb{R}^{m_\ell}$ be the input to layer ℓ and, given some index set $J \subseteq [m_\ell] = \{1, \dots, m_\ell\}$, let $\phi_J(x) = [\phi_j]_{j \in J} \in \mathbb{R}^{|J|}$ be a subvector of $\phi(x)$. The index set J is the collection of units, which in a fully connected layer is rows of the weight matrix. Our goal is to obtain the most useful information from $\phi(x)$ and describe it in a smaller output. Finding this optimal representation can be formulated as the following optimization problem

$$A_J = \underset{A \in \mathbb{R}^{m_\ell \times |J|}}{\arg\min} \; \hat{\mathbb{E}}\left[\|\phi - A\phi_J\|^2\right] + \|A\|_w^2 \tag{4.1.1}$$

where $\|A\|_w^2 = \mathbf{tr}[A I_w A^T]$ is a regularizing term, $I_w = diag(w)$ and $\hat{\mathbb{E}}(f) = \frac{1}{n}\sum_{i=1}^n f(x_i)$ is the empirical expectation with respect to the dataset. Also let $\hat{\Sigma} := \hat{\mathbb{E}}[\phi\phi^T]$ be the empirical covariance matrix, $\hat{\Sigma}_{I,J} := \hat{\mathbb{E}}[\phi_I \phi_J^T] \in \mathbb{R}^{|I| \times |J|}$ be the submatrix given by the indexes I and J, and $F = [m_\ell] \backslash J$ be the corresponding complement of J. We also assume that $\hat{\Sigma}_\ell$ and all its submatricies are positive definite and therefore non singular. We can now present the known solution for (4.1.1)

$$A_J = \hat{\Sigma}_{F,J}(\hat{\Sigma}_{J,J} + I_w)^{-1} \tag{4.1.2}$$

The method combines an output and input aware method. This makes it possible to find the best combination of the two methods. By including the solution (4.1.2) and using the spectral norm, $\|A\|_{op} = \max_i |\mu_{A,i}|$ the maximal eigenvalue, we can reformulate the objective function. The input aware method is defined as

$$L_w^{(A)}(J) = \left\|\hat{\Sigma}_{F,F} - \hat{\Sigma}_{F,J}(\hat{\Sigma}_{J,J} + I_w)^{-1}\hat{\Sigma}_{J,F}\right\|_{op} \tag{4.1.3}$$

Then we need to consider at set of vectors $\mathcal{Z}_\ell \subset \mathbb{R}^{m_\ell}$ which approximates the output $z^T\phi$ for $z \in \mathcal{Z}_\ell$ well. We then construct a matrix Z_ℓ which either consists of rows of the elements of $\mathcal{Z}_\ell$ if $\mathcal{Z}_\ell$ is finite or the projection matrix onto the span of $\mathcal{Z}_\ell$ if $\mathcal{Z}_\ell$ is infinite. A practical approach is to simply use the trained weight matrix $Z_\ell = W_\ell$. This lets us define the output aware method

$$L_w^{(B)}(J) = \left\|Z_\ell[\hat{\Sigma}_{F,F} - \hat{\Sigma}_{F,J}(\hat{\Sigma}_{J,J} + I_w)^{-1}\hat{\Sigma}_{J,F}]Z_\ell^T\right\|_{op} \tag{4.1.4}$$

and the convex combination with parameter $0 \leq \theta \leq 1$

$$L_w^{(\theta)}(J) = \theta L_w^{(A)}(J) + (1-\theta)L_w^{(B)}(J). \tag{4.1.5}$$

By choosing θ we obtain the following optimization problem

$$\min_{J\subset[m_\ell]} L_w^{(\theta)}(J). \tag{4.1.6}$$

This is however hard to compute. We can simplify the problem by replacing the spectral norm with trace and setting $w = 0$. Then by choosing some level $\alpha > 0$ we obtain a new formulation

$$\min_{J\subset[m_\ell]} |J| \ \text{ such that } \ \mathcal{C}(J) = \frac{\mathbf{tr}((\theta I - (1-\theta)R_z)\hat{\Sigma}_{F,J}\hat{\Sigma}_{J,J}^{-1}\hat{\Sigma}_{J,F})}{\mathbf{tr}((\theta I - (1-\theta)R_z)\hat{\Sigma}_{F,F})} \geq \alpha. \tag{4.1.7}$$

where $R_z = Z_\ell^T Z_\ell$.

This is computationally demanding and can be simplified by using a greedy method. We start with an empty $J = \emptyset$ and then update $J \leftarrow \arg\max_{j\in F} \mathcal{C}(J \cup \{j\})$. We do this iteration until $C(J) \geq \alpha$ or alternatively $|J| = \hat{m}_\ell$ if we have an estimate of the layer width, $\hat{m}_\ell$. Specifically we need to compute $(\hat{m}_\ell + 1)(m_\ell - \frac{\hat{m}_\ell}{2})$ evaluations of $\mathcal{C}(J)$, where $\hat{m}_\ell$ is the size of the set outputted from the optimization. For example a modestly designed network where $m_\ell = 1000$ and $\hat{m}_\ell = 500$ will still require 375 750 evaluations and the evaluation of $\mathcal{C}(J)$ can sometimes take several seconds on a normal CPU.

4.2 Estimating the layer width

We can also try to solve an easier problem by using the degree of freedom. The degree of freedom is defined as

$$N_\ell(\lambda_\ell) = \mathbf{tr}\left[\hat{\Sigma}_\ell(\hat{\Sigma}_\ell + \lambda_\ell I)^{-1}\right] = \sum_{i=1}^{m_\ell} \frac{\mu_{\ell,i}}{\mu_{\ell,i} + \lambda_\ell} \tag{4.2.1}$$

where $\{\mu_{\ell,i}\}_{i=1}^{m_\ell}$ are the eigenvalues of $\hat{\Sigma}_\ell$. This can serve as an approximation to the layer width $\hat{m}_\ell \approx N_\ell(\lambda_\ell)$ and it is much easier to compute than (4.1.7). The problem is that this only gives us the size of the layers not the elements it should contain.

The problem of compressing can be interpreted as minimizing the generalization error which gives us the objective function

$$F(\lambda) = \sum_{\ell=1}^{L} \hat{m}_\ell \hat{m}_{\ell+1} \tag{4.2.2}$$

where $\lambda = [\lambda_1, \dots, \lambda_L]$. We need to require that $\lambda_\ell \geq 0$. This can be solved by changing the variable to $p_\ell^2 = \lambda_\ell$ and adding a regularizing term with parameter ρ which gives

$$\min_{p\in\mathbb{R}^L} \sum_\ell N_\ell(p_\ell^2)N_{\ell+1}(p_{\ell+1}^2) + \rho R(p_\ell). \tag{4.2.3}$$

where is R a regularizer, for example $R(p_\ell) = \sum_\ell |p_\ell|$. Next we need to configure ρ such that it gives a good estimate for the layer width.

4.3 Experiments

The experimental process can be divided into different parts.

1. A network, $f(x)$, of choice is trained on a specific dataset.
2. The empirical covariance $\hat{\Sigma}_\ell$ is computed for for every layer.
3. The SVD (eigenvalues) $(\{\mu_k\}, \{v_k\})_\ell$ of $\hat{\Sigma}_\ell$ is computed.

4. We can now solve problem (4.2.3) and find a suitable value for ρ. We get an estimate of the optimal layer width $\hat{m}_\ell$.
5. We compute the best subset of parameters given the new architecture $(\hat{m}_\ell)_{\ell=0}^{L-1}$.
6. The new network is tested for performance both initially and after running additional training epochs (fine tuning).

To test our method we have generated a small dataset which we will call MINI. It is generated as follows. Sample $T \sim \mathcal{N}_{30}(0, I + \varepsilon \mathbb{1}\mathbb{1}^T)$ with $\varepsilon = 0.1$, then compute

$$X = \begin{cases} \exp(T) & \text{if } \|T\|_2 < 1 \\ \log(|T|) & \text{if } \|T\|_2 \geq 1 \end{cases} \qquad Y = \begin{cases} 0 & \text{if } \bar{X} < -0.6 \\ 1 & \text{if } \bar{X} \geq -0.6 \end{cases} \tag{4.3.1}$$

and then let MINI $= \{(X, Y)\}$ be our dataset which we can partition into a training set and a test set. Then we trained a small NN, which we will call MiniModel, on MINI. MiniModel consists of four dense layers, with or without biases, with initial output dimension (widths) 16, 12, 10, and 1. This means that the model has 802 parameters (without biases) which is much more than samples in the training set (100). We then trained on MINI for 200 epochs. This gave us a good training loss, but not a good test loss, which means it is not generalizing well. It also did not manage to learn the dataset completely (get 0 loss) which might indicate that we did not have enough parameters, data or run enough training epochs.

We proceeded with the experiment and computed the empirical covariance and corresponding eigenvalues. Then we did step 4 for a range of ρ which can be seen in figure 1. Note that the final layer is not included since its size is determined by the output dimension. Then we computed the new best index set as given by (4.1.7) and created a new network with the new dimensions. We observed an immediate improvement to the test loss and even better after fine tuning. The result of which can be seen in table 1.

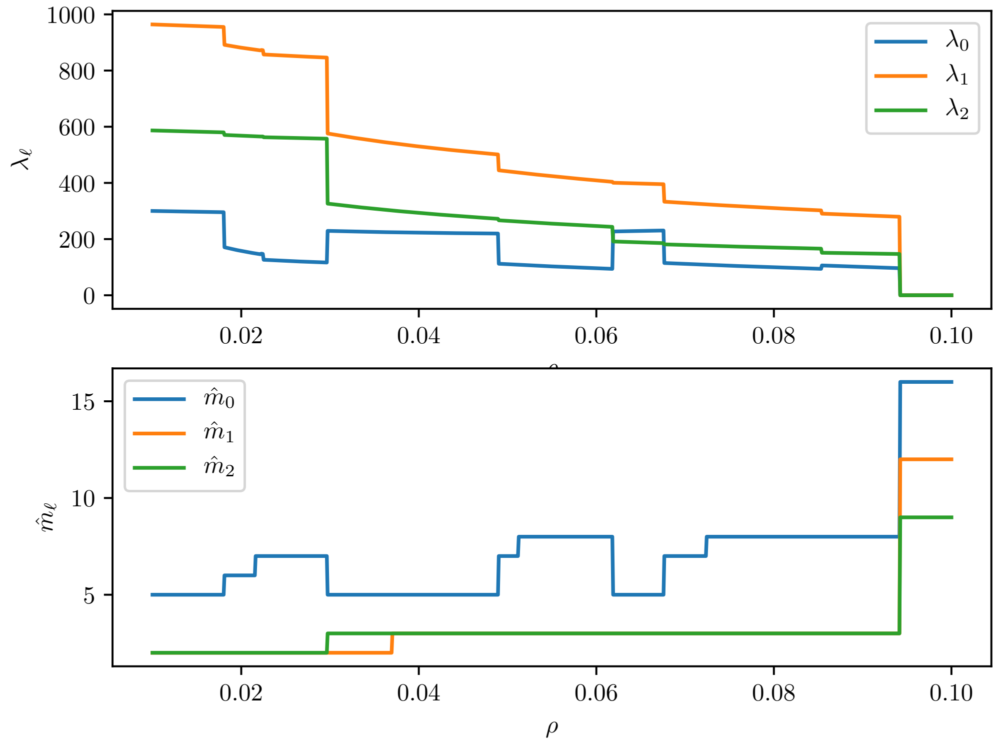

Figure 1: How the optimization (4.2.3) behaves on MiniModel for different values of $\rho \in [0.01, 0.1]$. When passing the point $\rho \approx 0.09$ the optimization returns the original layer width since $\lambda = \boldsymbol{0}$. From this plot we select the architecture with layer widths $(8, 3, 3)$.

	Original	**Compressed**	**Fine tuned**
Test loss	1.45	0.90	0.87
Accuracy	0.52	0.44	0.60

Table 1: The test score of MiniModel before compression of to $(8, 3, 3)$ layer widths.

5 Discussion

First we will discuss the potential of spectral pruning as a method for selecting, or rather improving, the architecture of a NN. We wanted to investigate how the estimated layer width from (4.2.3) behaves with respect to different ρ on a CNN trained on the CIFAR-10 dataset. The CNN consists of 4 convolutional layers with a max pooling layer in between every other and at the end two fully connected layers.

To compute the empirical covariance for these layers is however very expensive. For example the output of the first convolutional layer is of shape $32 \times 32 \times 32$ which means that the flattened covariance matrix will be of size $32^3 \times 32^3$ which contains roughly $\sim 10^9$ elements. This was however too computationally expensive to be achieved with the time and resources available.

We have shown that spectral pruning can be applied to a network and compress it to achieve better performance. However it might fall short of the performance and capabilities of ENAS and NASNet.

The computational demand is also an issue. If this technique is going to be applied with success to practical problems further approximations might be necessary. However insights from this method might be used to improve other methods and vice versa.

The no free lunch theorem shows that every algorithm has limitations which means that NNs also will fail to learn most tasks. It would be interesting to discover for which problem classes there exists NN architectures that can learn the task on a competitive level to other methods like Support Vector Machines (SVMs). It is already known that conventional NNs are outperformed by other algorithms on many simple problems. Additionally, despite the name, NNs are far from representing biological NNs as found in human or animal brains. It is possible that further investigation of the biological NNs can uncover new useful techniques for deep learning.

We have argued in this project thesis that mathematical analysis and insight is essential in the further development deep learning. This means that the project of developing powerful machine learning algorithms and artificial intelligence can benefit immensely by including mathematicians and statisticians.

6 Further work

There is a need for better tools to analyze deep learning methods. If we can combine some of the insights and theories in this thesis we can potentially discover new and better techniques in deep learning. It may be necessary for mathematical analysis to catch up to the practical progress in order to maintain security and trustworthiness of deep learning methods.

Outside of improving application areas that are already somewhat successful, like image classification and automatic translation, a better understanding of the limitations and potential of deep learning can give indications of where efforts to application are worthwhile. If one could construct an architecture space that could encompass as many learning algorithms as possible one could potentially search this space for the best architecture for a particular problem.

One direction for further development of spectral pruning is to see if it can be used to improve any of the automatic model design methods. For example if the covariance matrix eigenvalues are small it indicates redundancy in the layer and might be used as some sort of penalty in the architecture gradient.

7 Conclusion

We have presented a large body of research in deep learning and connected and discussed the theory of those articles. This creates a basis for further study and development of deep learning. The reader can take this as an overview and inspiration to further research.

References

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[BR88] Avrim Blum and Ronald L. Rivest. Training a 3-node neural network is np-complete. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, NIPS'88, pages 494–501, Cambridge, MA, USA, 1988. MIT Press.

[CT06] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, New York, NY, USA, 2006.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GPAM$^+$14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, page 2672–2680. Curran Associates, Inc., 2014.

[GS18] William H. Guss and Ruslan Salakhutdinov. On characterizing the capacity of neural networks using algebraic topology. *CoRR*, abs/1802.04443, 2018.

[Hat02] A. Hatcher. *Algebraic Topology*. Algebraic Topology. Cambridge University Press, 2002.

[KGS$^+$17] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit. One Model To Learn Them All. *ArXiv e-prints*, June 2017.

[Li17] Y. Li. Deep Reinforcement Learning: An Overview. *ArXiv e-prints*, January 2017.

[MT17] Michal Moshkovitz and Naftali Tishby. Mixing complexity and its applications to neural networks. *CoRR*, abs/1703.00729, 2017.

[NS18] Atsushi Nitanda and Taiji Suzuki. Gradient layer: Enhancing the convergence of adversarial training for generative models. In *AISTATS*, 2018.

[PGZ$^+$18] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018.

[RSG16] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016.

[SAM$^+$18] T. Suzuki, H. Abe, T. Murata, S. Horiuchi, K. Ito, T. Wachi, S. Hirai, M. Yukishima, and T. Nishimura. Spectral-Pruning: Compressing deep neural network via spectral analysis. *ArXiv e-prints*, August 2018.

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA, 2014.

[ST17] R. Shwartz-Ziv and N. Tishby. Opening the Black Box of Deep Neural Networks via Information. *ArXiv e-prints*, March 2017.

[TPB99] Naftali Tishby, Fernando Pereira, and William Bialek. The information bottleneck method. *CoRR*, physics/0004057, 1999.

[TSK$^+$18] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. *Lecture Notes in Computer Science*, page 270–279, 2018.

[WLT$^+$18] W. Wei, L. Liu, S. Truex, L. Yu, M. Emre Gursoy, and Y. Wu. Adversarial Examples in Deep Learning: Characterization and Divergence. *ArXiv e-prints*, June 2018.

[WM97] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Trans. Evolutionary Computation*, 1(1):67–82, 1997.

[ZC05] Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, Feb 2005.

[ZSX17] Guanhua Zheng, Jitao Sang, and Changsheng Xu. Understanding deep learning generalization by maximum entropy. *CoRR*, abs/1711.07758, 2017.

[ZVSL17] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017.

8 Appendix

8.1 Guide to notation

For convenience we include an additional list of some of the notation used in the text.

- $\|A\|_w^2 = \mathbf{tr}[AI_wA^T]$ w-norm
- $\hat{\mathbb{E}}(f) = \frac{1}{n}\sum_{i=1}^n f(x_i)$ empirical expectation
- $\|A\|_{op} = \max_i |\mu_{A,i}|$ spectral norm, the maximal singular value
- $\hat{\Sigma} := \hat{\mathbb{E}}[\phi(X)\phi(X)^T] = \frac{1}{n}\sum_{i=1}^n \phi(x_i)\phi(x_i)^T$ empirical covariance matrix
- $[m] = \{1, \ldots, m\}$ is a shorthand for an integer range.
- $\bar{X} = \frac{1}{n}\sum_i X_i$, where n is the dimension of X, is the mean of a random vector X.
- $\mathcal{N}_p(\mu, \Sigma)$ is a multivariate normal distribution with mean μ and covariance Σ of dimension p.
- Big O-notation $f(x) = O(g(x))$ if there exists $M > 0$ and x_0 such that $|f(x)| \leq Mg(x)$ for all $x \geq x_0$.
- Big Ω-notation $f(x) = \Omega(g(x))$ if and only if $g(x) = O(f(x))$.

8.2 Additional plots

Some additional plots that was not included in the main text are presented in this section.

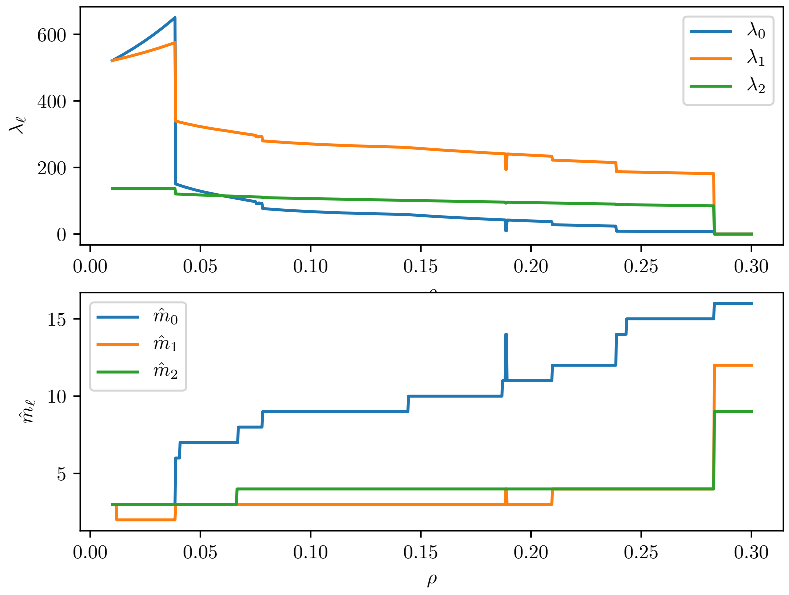

Figure 2: How the optimization (4.2.3) behaves on MiniModel with bias for different values of ρ.

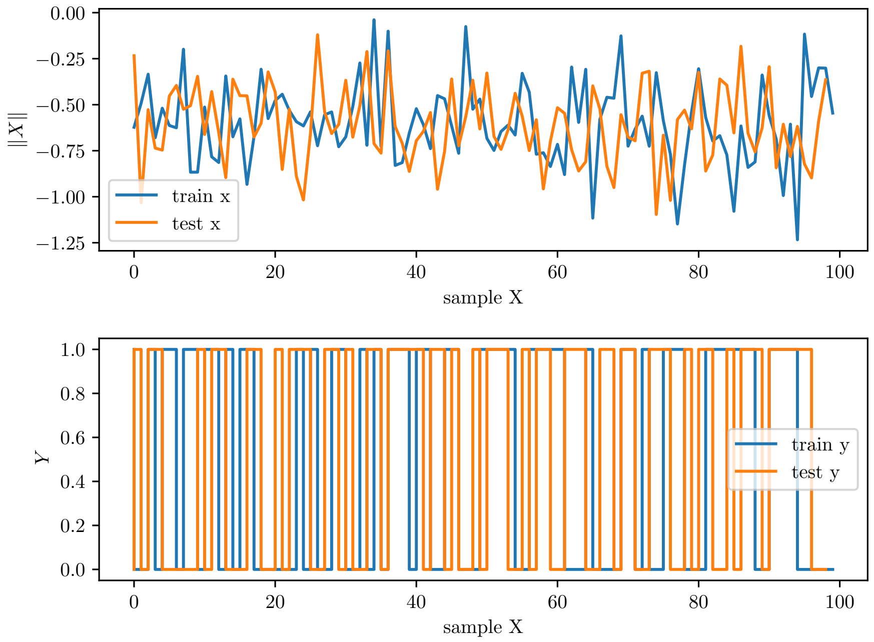

Figure 3: A plot of the MINI dataset.

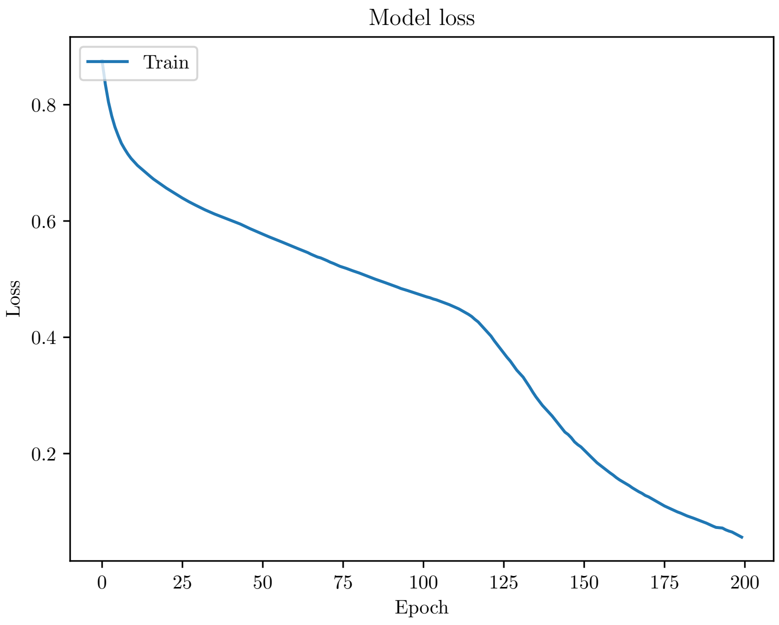

Figure 4: Training loss of MiniModel.

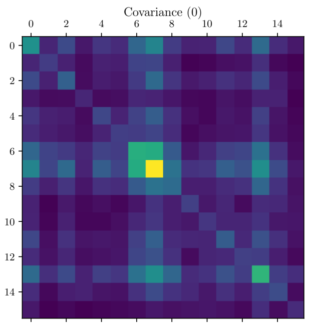

Figure 5: The empirical covariance matrix of the first layer for MiniModel.

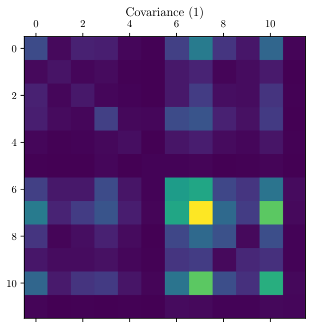

Figure 6: The empirical covariance matrix of the second layer for MiniModel.

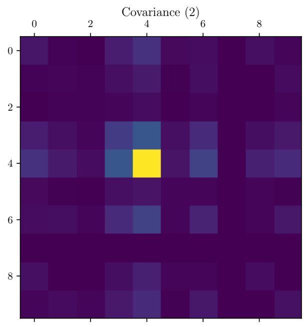

Figure 7: The empirical covariance matrix of the third layer for MiniModel.